31. Generic Refactoring for Programming and Modeling Languages

Jan Reimann, Mirko Seifert, Prof. Uwe Aßmann

Version 15-0.2, 17.1.11

Obligatory Literature

  • doi:10.1109/ISPSE.2000.913233,
• MOOSE framework http://www.moosetechnology.org/
References

- http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-177153

An Example of Code Refactoring

Extract Method (Outlining)
From Code to Models
Why is Refactoring needed for Models?

- Model-Driven Software Development:
  - Models are partial code
  - Models are primary artefacts in MDSD
  - Good model design is essential for understandability
  - Some models are domain-specific, and belong to domain-specific languages (DSL)

Why should it be generic?
- Known code refactorings are transferable to many DSLs
- Core steps of refactorings are equal for different metamodels
- A lot of additional effort to specify refactorings from scratch

---

**M0-M3 Metahierarchy**

- M3: Common Object-Oriented Meta-Metamodel
- M2: Target Metamodel
- M1: Adaptation
- M0: Based on
31.2 MOOSE

- Its common upper metamodel FAMIX
- How to refactor a common upper metamodel

http://www.moosetechnology.org/?_s=5k2-x-GDjidd2Y1X
The FAMIX upper metamodel

- Enables generic refactoring for all entities *above methods, not touching method bodies*, such as class restructurings, class renamings, package refactorings, etc.
- The MOOSE framework supplies basic graph algorithms for reengineering and refactoring:
  - Strongly connected components
  - Dominance
  - Kruskal spanning trees
- Concept recognition in texts
- Formal concept analysis

Common Upper Metamodels – Limitations

- The refactorer can be based on a common meta-metamodel, like in FAMIX
- Then, it is general, but limited to the upper metamodel
- Lack of exact control of structures to be refactored

[Moha, Naouel, Vincent Mahé, Olivier Barais und Jean-Marc Jézéquel: *Generic Model Refactorings*, MODELS 2009]
Related Work – Limitations
M2 layer specification

- If the refactorer is based on a specific M2 metamodel, there is no genericity

- No genericity
- No reuse

[Taentzer, Gabriele, Dirk Müller and Tom Mens: Specifying Domain-Specific Refactorings for AndroMDA Based on Graph Transformation, AGTIVE 2007]

Related Work – Limitations
M1 layer specification

- If the refactorer is based on M1 transformation examples, it is too inflexible

- No genericity
- No reuse

[Brosch, Petra, Philip Langer, Martina Seidl, Konrad Wieland, Manuel Wimmer, Gerti Kappel, Werner Retschitzegger and Wieland Schwinger: An Example is Worth a Thousand Words: Composite Operation Modeling By-Example, MODELS 2009]
31.2 Refactory – A Tool for Writing Refactorers

The generic refactorer of TU Dresden
Jan Reimann

Role-based Design (Reenskaug, Riehle & Gross)

- Definition of collaborations of objects in different contexts
- Here: Context = model refactoring
- Participants play role in concrete refactoring → Role Model
  - The refactorer is programmed against the role model
- Role-based transformation → Refactoring Specification
- Application to desired parts of metamodel → Role Mapping

Role-based Generic Model Refactoring

M2

<table>
<thead>
<tr>
<th>DSL Designer</th>
<th>DSL Meta Model</th>
<th>Role Mapping</th>
<th>Role Model</th>
<th>Refactoring Specification</th>
<th>Refactoring Designer</th>
</tr>
</thead>
</table>

M1

<table>
<thead>
<tr>
<th>DSL User</th>
<th>DSL Model</th>
<th>Refactoring Interpreter</th>
<th>Refactored DSL Model</th>
</tr>
</thead>
</table>

slides
• Refactory sees a role model (a view) of the metamodel

Refactoring Specification on Role Model

• The roles of this role-metamodel can be used to write refactoring scripts and operators
• The refactore consists of a set of refactoring scripts
A role-class **mapping** maps roles to metaclasses in a concrete metamodel.
Evaluation of Refactory

Starting point
- 17 target metamodels of different complexity (Java, UML, Ecore...)
- 76 concrete model refactoring

Result
- 11 generic model refactoring
- 6 metamodel specific extensions were needed (postprocessors)
- 10 metamodels are multiple target of same generic refactoring
- 2 metamodels are at least target of every generic refactoring

New: Multi-Quality Contracts in CPS (Multi-Technical Spaces)
New: Co-Refactoring of Multi-Quality Contracts in CPS (Multi-Technical Spaces)

Component A

Real-time

Safety

Dynamics

Energy

Refactorer 1

Real-time contract checking (Technical Space 1)

Refactorer 2

Safety contract checking (Technical Space 2)

Refactorer 3

Dynamics contract checking (Technical Space 3)

Refactorer 4

Energy contract checking (Technical Space 4)

Generic Refactorer

Component B

Folie 21 von XY

Folie 22 von XY
Lessons Learned

- Refactorings generically specifiable if abstractable and structurally transferable
- Metamodel-specific refactorings possible
- Design decisions
  - “Specific” generic refactoring
  - Metamodel-specific extension or
  - Implementation of metamodel-specific refactoring (Java)
- Reuse beneficial if model refactoring appliable to at least two metamodels
- Co-refactorers are possible

Contributions

- Generic refactoring works!!
- Definition of generic model refactorings based on roles
  - Role models form a dedicated context for every model refactoring
  - Approach allows both for genericity and control of the structures to be refactored
  - Control is achieved by mapping of role models into arbitrary sections of the target metamodel
  - Interpretation by resolving roles and collaborations into the target metamodel
Contributions

Open Issues

- Proof of behavior preservation with formalization of semantics
- Automatic mapping to metamodels
  - Is hard because of many possibilities for complex metamodels
  - But semi-automatic support with graph querying

http://www.modelrefactoring.org

jan.reimann@devboost.de
Uwe.assmann@tu-dresden.de
Mapping to Paths

![Diagram showing mapping between SubElement and SuperElement through Classifier and Generalization relationships.](image-url)